CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Notice

- HW2 posted on Blackboard, and will be due on this Sunday midnight.

Recap

- Course Introduction
- Java Review

Arrays

- An array is a dynamically-created object that serves as a container to hold constant number of values of the same type.
- The array is the most commonly used data structure, and it is built into most programming languages.
- Properties of the array data structure:
    - Elements in the array are allocated with contiguous memory.
    - With fixed length
    - Each element is associated with an index
    - Index starts with 0.

Arrays

- Declare and create an array object of a specified type.
  - int[] intArray = new int[10];
  - String[] strArray new String[20];
  - int[][] twoDArray = new int[5][5];
- Declare, create and initialize array element at the same time.
  - int[] intArray = {1, 3, 5, 7, 9};
  - String[] strArray = {“a”, “b”, “c”, “d”, “e”};
- Get the length of an array: intArray.length
- Access an element in an array: intArray[0], intArray[intArray.length-1];
- Traverse the entire array: for-loop, for-each loop
- Display all elements in an array

Basic Operations on arrays

- For any data containers (collections) as an array, we concern about the following operations:
  - Insert data: how to add a new element into this collection.
  - Delete data: how to delete an element from this collection.
  - Swap elements: how to swap two elements.
  - Search: search whether a given target element exists in the array.
  - Duplicate allowed?: we sometimes also care about whether duplicate values is allowed to exist in the collection.

Basic Operations on Arrays

- Insert
  - Easy to insert at the end, consider the size

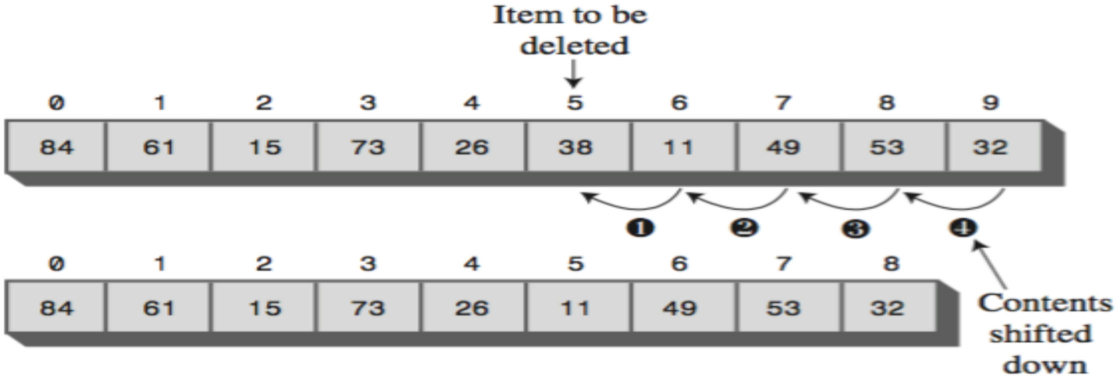
```
intArray[0] = 100 ;
```

- Swap two elements
  - Use a temporary variable

```
int temp = intArray[0];
intArray[0]= intArray[1];
intArray[1] = temp;
```

Basic Operations on Arrays

- Delete
  - Search the target, then shift down the rest of the array elements.



0	1	2	3	4	5	6	7	8	9
84	61	15	73	26	38	11	49	53	32

0	1	2	3	4	5	6	7	8
84	61	15	73	26	11	49	53	32

```
for(int i=0; i<intArray.length; i++) {
        if (target==intArray[i]) {
                     for(int j=i; j<intArray.length; j++) {
                        intArray[j]=intArray[j+1];
                     }
              }
}
```

Basic Operations on Arrays

- Search
  - Go over each element until target found (Linear Search)

```
for(int i=0; i<intArray.length; i++)
{
        if (target==intArray[i])
                {
                        targetIndex=i;
                }
}
```

Cost of Operations on Arrays

- For now, lets consider number of comparisons and moves (on average).

	Insert	Search	Delete
# of Moves	1	0	n/2
# of Comparisons	0	n/2	n/2

A Class for Array

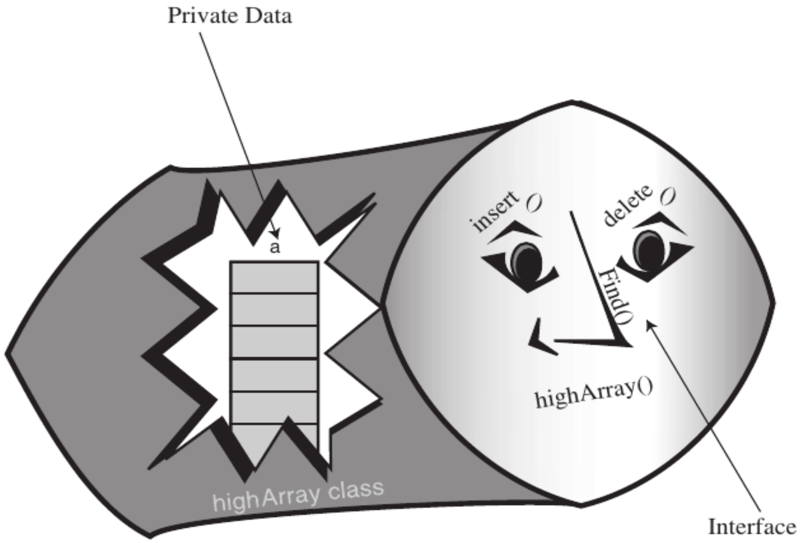


FIGURE 2.4 The `HighArray` interface.

```
class HighArray {
private int[] intArray;
private int Nelems;
//----------- Constructor
public HighArray(int max)
{
intArray = new int[max];
Nelems =0;
}
//------------Methods
public void insert(int value)
{
intArray[Nelems]=value;
Nelems++;
}
public bolean find(int value)
...
public bolean delete(int value)
...
public void display()
...
}
```

A Class for Array

- Focus on *What Vs How*
  - My application class does not have to worry about implementation details.
  - It does not even know the data structure.
    - HighArray array = new HighArray(10);
    - array.delete(500);
    - array.insert(122);
    - System.out.println(array.find(122));
    - array.display();

Ordered Array

- Suppose we are implementing another data structure as “Ordered Array” in which all elements are stored in ascending order
- We also concern about the basic operations: insert, delete, search. What will it be different from unordered array.
  - Insertion is costly. Why?
    - You have to find the right place and shift elements
  - However, it can enable a faster search method
- Good idea to keep the order when you expect to do more search operations than insert/delete.

Binary Search

- Analogous to **guess-a-number** game.
  - You want to guess a number I have in mind. The number is between 0 and 100 and after each guess, I tell you if your guess is too high or too low.

Step	Number Guessed	Result	Range of Possible Values
0			1–100
1	50	Too high	1–49
2	25	Too low	26–49
3	37	Too high	26–36
4	31	Too low	32–36
5	34	Too high	32–33
6	32	Too low	33–33
7	33	Correct	

Binary Search Implementation

```
public int find(int searchKey) {
int lowerBound = 0;
int upperBound = nElems-1;
int curIn;
while(true) {
  curIn = (lowerBound + upperBound) / 2;
  if(a[curIn]==searchKey)
      return curIn; // found it
else if(lowerBound > upperBound)
      return nElems; // not found
else // divide range {
  if(a[curIn] < searchKey)
     lowerBound = curIn + 1; //in upper half
 else
    upperBound = curIn - 1; //in lower half
} // end else divide range
} // end while
} // end find()
```

How Fast is Binary Search?

- Let’s find the number of steps needed to find the element
  - At each step, we eliminate half of the search space
  - We continue until one element is left (worst case)
  - Example, n=100 elements

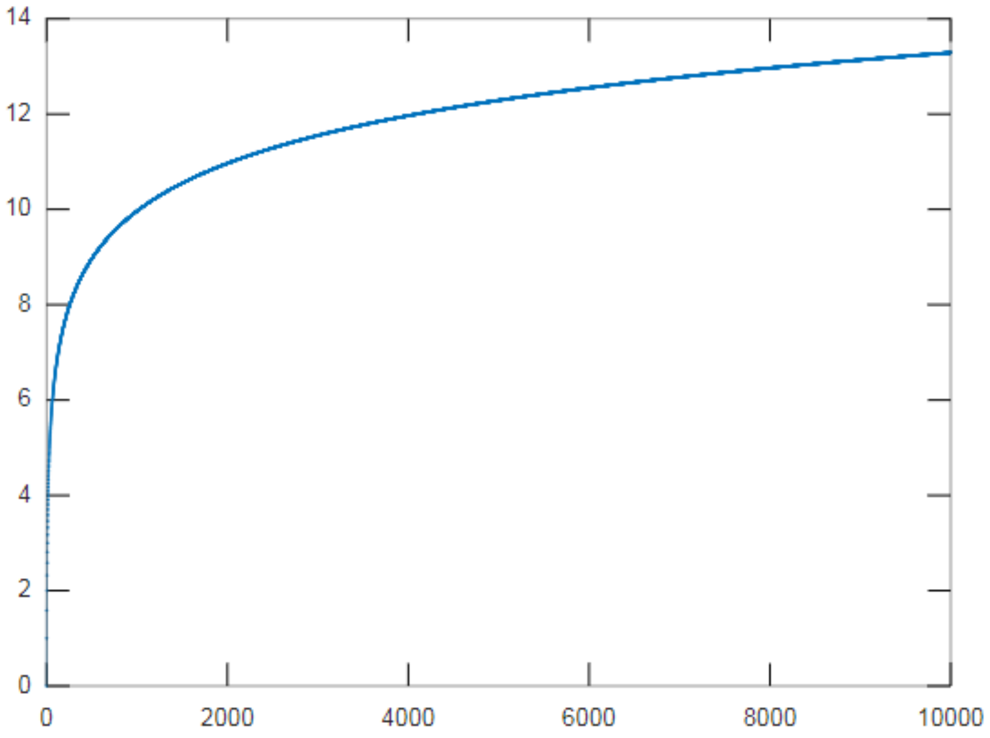
Step	n		Number of items
1	100/2 = 50.0000	$(n/2^{1})$	50
2	50/2 = 25.0000	$(n/2^{2})$	25
3	25/2 = 12.5000	$(n/2^{3})$	13
4	12.5/2 = 6.25000	$(n/2^{4})$	7
5	6.25/2 = 3.12500	$(n/2^{5})$	4
6	3.125/2 = 1.56250	$(n/2^{6})$	2
7	1.5625/2 = 0.78125	$(n/2^{7})$	**1**

We at most need to guess **7** times to successfully find out the secret number.

How Fast is Binary Search?

- Let *x* be the number of steps needed to find the element in the worst case (having one element left).
  - $n \,/\, 2^x = 1$ *or* $n = 2^x$

$$\boxed{x = \log_2(n)}$$



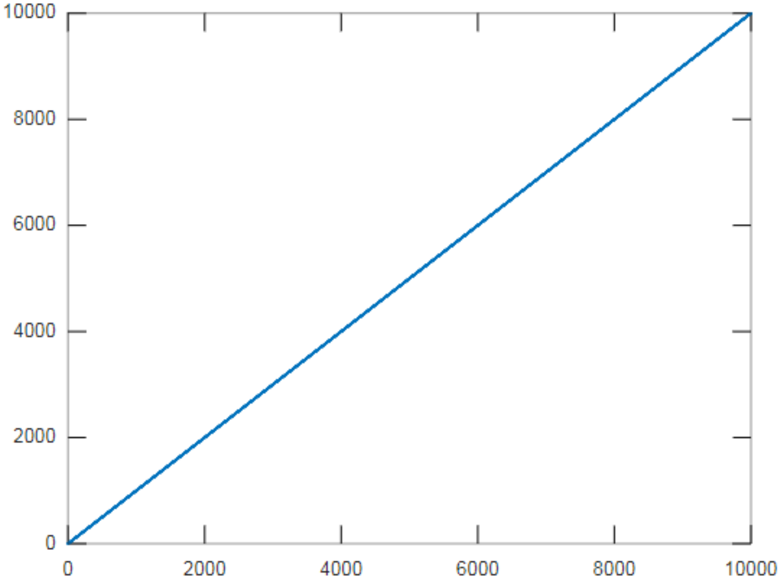
Linear Search Vs Binary Search?

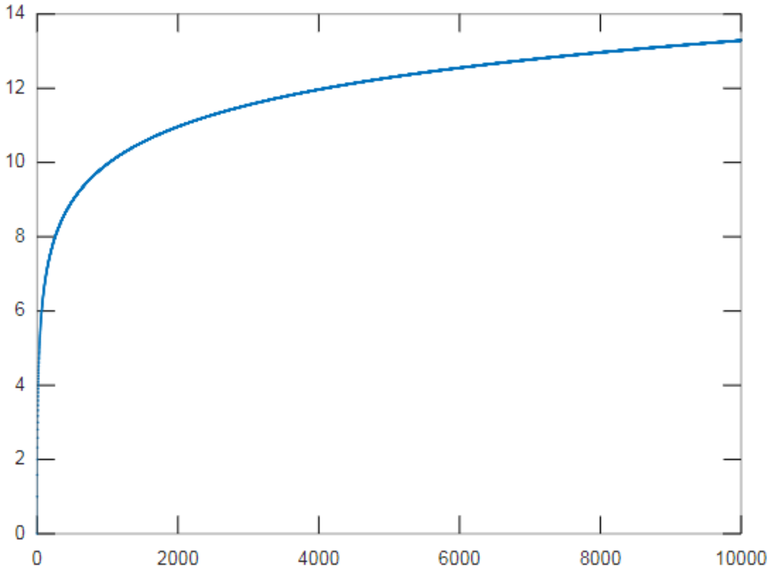
n	**Linear search**	**Binary search**
10	10	4
100	100	7
1,000	1,000	10
10,000	10,000	14
100,000	100,000	17
1,000,000	1,000,000	20
10,000,000	10,000,000	24
100,000,000	100,000,000	27
1,000,000,000	1,000,000,000	**30**

Notice the difference between linear search and binary search.
For very small numbers, the difference is not dramatic.
However, the more items there are, the bigger the difference.

We say that for all but very small inputs, the binary search is greatly superior.

Linear Growth Vs Logarithmic Growth


10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000


14
12
10
8
6
4
2
0
0
2000
4000
6000
8000
10000